PATENT APPLICATION COVER SHEET
Attorney Docket No. 1503.68591

I hereby certify that this paper is being deposited with the United States Postal Service as EXPRESS MAIL in an envelope addressed to: Mail Stop PATENT APPLICATION, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on this date.

Oct. 24, 2003
Date

Express Mail Label No.: EV032735309US

PARALLEL PROCESSING METHOD FOR INVERSE MATRIX FOR SHARED MEMORY TYPE SCALAR PARALLEL COMPUTER

INVENTOR:

Makoto NAKANISHI

GREER, BURNS & CRAIN, LTD.
300 South Wacker Drive
Suite 2500
Chicago, Illinois 60606
Telephone: 312.360.0080
Facsimile: 312.360.9315
CUSTOMER NO. 24978

APPLICATION FOR

UNITED STATES LETTERS PATENT

SPECIFICATION

Inventor(s): Makoto NAKANISHI

Title of the Invention: PARALLEL PROCESSING METHOD FOR INVERSE MATRIX FOR SHARED MEMORY TYPE SCALAR PARALLEL COMPUTER

PARALLEL PROCESSING METHOD FOR INVERSE MATRIX FOR SHARED MEMORY TYPE SCALAR PARALLEL COMPUTER

Cross Reference

This patent application is a continuation in part application of the previous U.S. patent application, titled "PARALLEL PROCESSING METHOD FOR INVERSE MATRIX FOR SHARED MEMORY TYPE SCALAR PARALLEL COMPUTER", filed on June 11, 2002, application No. 10/288,984, herein incorporated by reference.

Background of the Invention

Field of the Invention

The present invention relates to an arithmetic process using a shared memory type scalar parallel computer.

Description of the Related Art

Conventionally, when an inverse matrix of a matrix is obtained using a vector computer, a computing method based on the Gauss-Jordan method and that operations are quickly performed by utilizing fast-access memory. For example, the methods such as a double unrolling method, etc. in

which an instruction in a loop is unrolled into a single list of instructions and executed once.

Described below is the method of obtaining an inverse matrix in the Gauss-Jordan method (or referred to simply as the Gauss method). (In the explanation below, the interchange of pivots is omitted, but a process of interchanging row vectors for the interchange of pivots is actually performed).

Assume that A indicates a matrix for which an inverse matrix should be calculated, and x and y indicate arbitrary column vectors. Ax = y is expressed as simultaneous linear equations as follows when matrix elements are explicitly written.

$$
\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = y_1 \\
&a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = y_2 \\
&\qquad \cdots\cdots\cdots \\
&a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = y_2
\end{aligned}
$$

If the above listed equations are transformed into By = x, then B is an inverse matrix of A, thereby obtaining an inverse matrix.

1) The equation in the first row is divided by a_{11}.

2) Compute the i-th row (i > 1) - the first row x a_{i1}.

3) To obtain the coefficient of 1 for x2 of the equation in the second row, multiply the second row by a reciprocal of the coefficient of x2.

4) Compute the i-th row (i > 2) - the second row x a_{i2}.

5) The above mentioned operation is continued up to the (n-1)th row.

The interchange of column vectors accompanying the interchange of pivots is described below.

Both sides of Ax = y are multiplied by the matrix P corresponding to the interchange of pivots.

$$PAx = Py = z$$

When the following equation holds with the matrix B,

$$x = Bz$$

then B is expressed as follows.

$$B = (PA)^{-1} = A^{-1}p^{-1}$$

That is, by right-multiplying the obtained B by P, an inverse matrix of A can be obtained. Actually, it is necessary to interchange the column vectors.

In the equation, $P = P_nP_{n-1} \ldots P_1$, and P_n is an orthogonal transform having matrix elements of $P_{ii} = 0$, $P_{ij} = 1$, $P_{jj} = 0$, and $P_{ji} = 1$.

In the vector computer, an inverse matrix is

computed in the above mentioned method based on the assumption that the operation of the memory access system can be quickly performed. However, in the case of the shared memory type scalar computer, the frequency of accessing shared memory increases with an increasing size of a matrix to be computed, thereby largely suppressing the performance of a computer. Therefore, it is necessary to perform the above mentioned matrix computation by utilizing the fast-accessing cache memory provided for each processor of the shared memory type scalar computer. That is, since the shared memory is frequently accessed if computation is performed on each row or column of a matrix, it is necessary to use an algorithm of localizing the computation assigned to processors by dividing the matrix into blocks, each processor processing the largest possible amount of data stored in the cache memory, and then accessing the shared memory, thereby reducing the frequency of accessing the shared memory.

Summary of the Invention

The present invention aims at providing a method of computing an inverse matrix at a high speed in a parallel process.

The method according to the present invention is a parallel processing method for an inverse matrix for shared memory type scalar parallel computer, and includes the steps of: specifying a predetermined square block in a matrix for which an inverse matrix is to be obtained; decomposing the matrix into upper left, left side, lower left, upper, lower, upper right, right side, and lower right blocks surrounding the square block positioned in the center; dividing each of the decomposed blocks into the number of processors and LU-decomposing the square block and the lower, right side, and lower right blocks in parallel; updating the left side, upper, lower, and right side blocks in parallel in a recursive program, and further updating in parallel using the blocks updated in the recursive program on the upper left, lower left, upper right, and lower right blocks; updating a predetermined square block in plural stages using one processor; and setting the position of the square block such that it can sequentially move on the diagonal line of the matrix, and obtaining an inverse matrix of the matrix by repeating the above mentioned steps.

According to the present invention, the

arithmetic operation of an inverse matrix is an updating process performed on each block, and each block is updated in parallel in a plurality of processors. Thus, the shared memory is accessed after the largest possible amount of operations are performed on a block stored in the cache provided for each processor, thereby realizing a localizing algorithm and performing a high-speed arithmetic operation for an inverse matrix.

Brief Description of the Drawings

FIG. 1 shows the configuration of the hardware of the shared memory type scalar computer according to an embodiment of the present invention;

FIG. 2 is a diagram (1) showing the order of computation according to an embodiment of the present invention;

FIG. 3 is a diagram (2) showing the order of computation according to an embodiment of the present invention;

FIG. 4 is a diagram (3) showing the order of computation according to an embodiment of the present invention;

FIG. 5 is a diagram (4) showing the order of computation according to an embodiment of the

present invention;

FIG. 6 is a diagram (5) showing the order of computation according to an embodiment of the present invention;

FIGS. 7A through 7F are diagrams (6) showing the order of computation according to an embodiment of the present invention;

FIGS. 8A through 8C are diagrams (7) showing the order of computation according to an embodiment of the present invention;

FIG. 9 shows a pseudo code (1) according to an embodiment of the present invention;

FIG. 10 shows a pseudo code (2) according to an embodiment of the present invention;

FIG. 11 shows a pseudo code (3) according to an embodiment of the present invention;

FIG. 12 shows a pseudo code (4) according to an embodiment of the present invention;

FIG. 13 shows a pseudo code (5) according to an embodiment of the present invention;

FIG. 14 shows a pseudo code (6) according to an embodiment of the present invention;

FIG. 15 shows a pseudo code (7) according to an embodiment of the present invention.

FIG. 16 is a flowchart (1) of the process of

the pseudo code;

FIG. 17 is a flowchart (2) of the process of the pseudo code;

FIG. 18 is a flowchart (3) of the process of the pseudo code;

FIG. 19 is a flowchart (4) of the process of the pseudo code;

FIG. 20 is a flowchart (5) of the process of the pseudo code;

FIG. 21 is a flowchart (6) of the process of the pseudo code;

FIG. 22 is a flowchart (7) of the process of the pseudo code;

FIG. 23 is a flowchart (8) of the process of the pseudo code;

FIG. 24 is a flowchart (9) of the process of the pseudo code;

FIG. 25 is a flowchart (10) of the process of the pseudo code;

FIG. 26 is a flowchart (11) of the process of the pseudo code;

FIG. 27 is a flowchart (12) of the process of the pseudo code;

FIG. 28 is a flowchart (13) of the process of the pseudo code; and

FIG. 29 is a flowchart (14) of the process of the pseudo code.

Description of the Preferred Embodiments

The present embodiment provides an algorithm of obtaining an inverse matrix of a given matrix. In the shared memory type scalar computer, it is necessary to increase a ratio of computation to load. Therefore, according to an embodiment of the present invention, a method of dividing a matrix into blocks as matrix products so that an arithmetic operation can be efficiently performed. Furthermore, the computation density is enhanced by updating part of the matrix with a recursively calculated product of matrices of varying dimensions.

FIG. 1 shows the configuration of the hardware of the shared memory type scalar computer according to an embodiment of the present invention.

Processors 10-1 through 10-n have primary cache memory, and the primary cache memory can also be incorporated into a processor. Each of the processors 10-1 through 10-n is provided with secondary cache memory 13-1 through 13-n, and the secondary cache memory 13-1 through 13-n are

connected to an interconnection network 12. The interconnection network 12 is provided with memory modules 11-1 through 11-n which are shared memory. The processors 10-1 through 10-n read necessary data for arithmetic operations from the memory modules 11-1 through 11-n, store the data in the secondary cache memory 13-1 through 13-n or the primary cache memory through the interconnection network 12, and perform arithmetic operations.

In this case, reading data from the memory modules 11-1 through 11-n to the secondary cache memory 13-1 through 13-n or the primary cache memory, and writing the computed data from the primary cache memory from the secondary cache memory 13-1 through 13-n or the primary cache memory to the memory modules 11-1 through 11-n are performed much more slowly than the operation speed of the processors 10-1 through 10-n. Therefore, if these writing and reading operations are frequently performed, then the performance of the entire computer is badly lowered.

Therefore, to maintain high performance of the entire computer, an algorithm of reducing the access to the memory modules 11-1 through 11-n and performing the largest possible amount of

computation in a local system formed by the secondary cache memory 13-1 through 13-n, the primary cache memory, and the processors 10-1 through 10-n is required.

Therefore, according to an embodiment of the present invention, computation for an inverse matrix is performed as follows.

Only a rightward updating process is performed on a block having a certain block width in a matrix to be computed so that the information used in the updating process outside a block can be stored. That is, to update the remaining portion in the outer product of the row vector orthogonal to the column vector selected as the center in the past process is performed in the Gauss-Jordan method described above by referring to the conventional technology. That is, the method 2) of the above mentioned Gauss-Jordan is explicitly written as follows.

$$a_{ij} - c_{1j} \times a_{i1} \text{ (where } i, j > 1 : c_{1j} = a_{1j} / a_{11})$$

This equation means that by selecting the column vector in the first column as the center and selecting the row vector in the first row

orthogonal to the selected vector, the matrix formed by the second and subsequent rows x the second and subsequent columns is updated by the outer product of the column vector in the first column and the row vector in the first row.

Furthermore, Ax = y can be rewritten using I as a unit matrix into Ax = Iy, and when A is converted in the Gauss-Jordan method, the unit matrix in the right side is also converted.

In updating the unit matrix, the element (i, i) of the column corresponding to the coefficient deleted in A is $1/a_{ii}$ in the right matrix, and the number of the other elements in the right matrix is obtained by multiplying the elements of the column vector deleted in A by $1/a_{ii}$ and then by -1. In a column block matrix, left elements are also to be deleted.

FIGS. 2 through 8 show the order of computation according to an embodiment of the present invention.

The order of computation is described below.

First, in FIG. 2, the matrix M is divided into blocks of column block matrices from the left side, and the updating process is performed in the above mentioned method.

1) E and H are LU decomposed (refer to Patent Application No. Hei-12-358232).

2) B is updated by $B \leftarrow BU^{-1}$ using the upper triangular portion U of E.

3) $D \leftarrow L^{-1}D$ and $F \leftarrow L^{-1}D$ using the lower triangular portion of E.

4) A, C, G, and I are updated.

$A \leftarrow A - B \times D$, $C \leftarrow C - B \times F$, $G \leftarrow G - H \times D$, $I \leftarrow I - H \times F$

5) The upper triangular portion of E is updated. Since this portion has not been updated in the LU decomposition, it is updated at this timing.

6) The update of the upper portion of D and F is performed according to the information about D, E, and F.

At this time, the updating process is performed using a matrix product in the recursive program to enhance the computation density.

a) Recursive Program

The upper portions of D and F can be computed by the outer products of the respective row vectors and the column vector orthogonal to the row vector of E. To enhance the the density of computation, the following recursive program is used. The upper portions of D and F indicate, as described by

referring to FIG. 8, the upper portions of the row blocks of D and F. As clearly shown in FIG. 8, D and F can be updated from the upper portions of the row blocks.

```
      recursive subroutine rowupdate ()
      if (update width < 10) then
      The product of the matrix excluding the
diagonal element and the row block matrix is
subtracted from the upper triangular matrix of E
(refer to the explanation of FIG. 8 for details).
      else
      c The update width is divided into a first
half and a second half.
The first half is updated.
      c Then, the second half is updated after
calling rowupdate.
      call rowupdate ()
      return
      end
```

7) Then, while moving the elements on the diagonal line in the lower right direction, the rectangular portion in the column direction on the lower left with respect to the diagonal line is updated. As a result, the information required to update B and H is generated.

8) The potions on the right of B and H are updated. This process is performed as follows.

With d indicating a block width, the numbers are sequentially assigned from left to right starting with i, ..., i + d - 1.

A reciprocal of a_{ii} is obtained. Other portions are obtained by dividing each portion by a_{ii}, and an inverse sign is assigned.

The portion on the left of the (i + 1)th row is divided by $a_{i+1,i+1}$, and a reciprocal of $a_{i+1,i+1}$ is obtained.

The left portion is updated by multiplying the row i + 1 by the column i + 1 of the left portion, and then performing subtraction.

The above mentioned processes are repeated.

This portion is further divided into the following procedures. They also require the rearrangement into a recursive algorithm.

9) Finally, the rectangular portion in the row direction on the upper left portion of the element on the diagonal line of E is updated.

10) Finally performed is the process of interchanging the column vector into the inverse direction of the history of the interchange of the pivots.

The portion updated in 5) above is updated by E2 = E2 - a x b while sliding the diagonal elements of the diagonal-line portion (E2) shown in FIG. 3 along the diagonal line.

This portion has not been updated in the LU decomposition. After the update, the information about the upper triangular matrix required to update D and F can be obtained.

The portion updated in 7) above is updated by E3 = E3 - c x d while sliding the diagonal elements of the diagonal-line portion shown in FIG. 4.

Prior to the update, c is multiplied by the reciprocal of the diagonal element.

Consider that the diagonal element after the update is the reciprocal of the original diagonal element. The character d is considered to be obtained by multiplying each column element by a diagonal element before update, and then multiplying the result by -1.

As a result, the information about the lower triangular matrix required to update the portion on the left of B and H.

In 9) above, the updating process is performed by E1 = E1 - a x c while sliding the diagonal element of the diagonal-line portion shown in FIG.

5.

The 'a' after the update is considered to be obtained by multiplying the column element of a by the original diagonal element, and then by -1.

D) Details of the parallelism for the shared memory type scalar computer

0) The LU decomposition on E, F, H, and I is a parallel decomposition using the parallel algorithm of the LU decomposition.

1) Control of block width

An embodiment of the present invention is provided with a system of adjusting a block width depending on the scale of a problem and the number of processors used in the parallelism.

Since the block E is updated by one processor, the block width is determined to be ignored (about 1%) with consideration given to the total cost of the portion.

2) In the update in C 4) above, the updating processes by the respective matrix products are performed in parallel. Each processor equally divides the second dimension for parallel sharing in computation.

3) In the update in C 5), that is, the update of D and F is performed by each processor updating in

parallel the areas obtained by equally dividing the second dimension of D and F.

4) The update in C 8) above is performed in parallel by each processor by sharing the areas obtained by equally dividing the first dimension of H.

5) Finally, the interchanging process on column vectors is performed in parallel by equally dividing the first dimension of the entire matrix into areas.

The broken lines shown in FIG. 6 show an example of dividing an area in updating matrix portion in parallel.

- Access to memory in updating B or H

Described below is the case in which the recursive program operates up to the depth of 2.

FIGS. 7A, and 7B, 7C, and 7D, 7E, and 7F each is a set.

First updated is the diagonal-line portion shown in FIG. 7A. At this time, the diagonal-line portion and the dotted line triangular portion are used.

Refer to the pseudo code for details. Next, the diagonal-line portion shown in FIG. 7A is updated using the horizontal-line portion shown in

FIG. 7A and the rectangular diagonal-line portion shown in FIG. 7B. Then, the horizontal-line portion shown in FIG. 7A is updated using the triangular portion in bold lines.

Next, the diagonal-line portion shown in FIG. 7C is updated by the product of the right white-painted rectangle shown in FIG. 7C and the rectangle in bold lines shown in FIG. 7D.

Then, the diagonal-line portion shown in FIG. 7E is updated using the triangular portion in broken lines shown in FIG. 7F. Furthermore, the diagonal-line portion shown in FIG. 7E is updated by the product of the horizontal-line portion shown in FIG. 7E and the rectangle with diagonal lines shown in FIG. 7F. Finally, the horizontal-line portion is updated using the triangular portion in bold lines shown in FIG. 7F.

In the above mentioned procedure, reference to E is common. Therefore, E can be stored in the cache of each processor for reference. In addition, for example, reference to and update to B is processed in parallel on the areas in the row direction (areas obtained by dividing by bold broken lines).

- Memory access in updating D or F

Described below is the case in which the recursive program operates up to the depth of 2.

First updated is the left diagonal-line portion shown in FIG. 8A. At this time, the diagonal-line portion and the right triangular portion in broken lines shown in FIG. 8A are used.

Refer to the pseudo code for details. Then, the left diagonal-line portion shown in FIG. 8A is updated using the right rectangular portion with diagonal lines shown in FIG. 8A and the left vertical-line portion shown in FIG. 8A. Then, the left horizontal-line portion shown in FIG. 8A is updated using the right triangular portion in bold lines shown in FIG. 8A.

Next, the left diagonal-line portion shown in FIG. 8B is updated by the product of the right rectangle in bold lines shown in FIG. 8B and the left white painted rectangle shown in FIG. 8B.

Then, the left diagonal-line portion shown in FIG. 8C is updated using the right triangular portion in broken lines shown in FIG. 8C. The left diagonal-line portion shown in FIG. 8C is updated using the product of the right rectangle with diagonal lines shown in FIG. 8C and the left vertical-line portion shown in FIG. 8C. Finally,

the left vertical-line portion shown in FIG. 8C is updated using the right triangular portion in bold lines shown in FIG. 8C.

In the above mentioned procedure, reference to E is common. Therefore, E can be stored in the cache of each processor for reference. Furthermore, the reference to and update of D is processed in parallel on the areas in the row direction (areas obtained by dividing by bold broken lines).

FIGS. 9 through 15 show the pseudo code according to an embodiment of the present invention.

FIG. 9 shows the pseudo code of the main algorithm of the parallelism algorithm of an inverse matrix. In the following pseudo code, the row having C at the left end is a comment row. 'array a(k,n)' is an array storing the elements of the matrix whose inverse matrix is to be obtained. 'ip (n)' is an array storing the information used in interchanging rows in the LU decomposition subroutine. For the algorithm of the LU decomposition subroutine, refer to Japanese Patent Application Laid-open No.Hei-12-358232. 'nb' refers to the number of blocks specified when the LU decomposition is performed.

When the LU decomposition is completed on one

specified block, and if ip(i) is larger than i, then the i-th row of the matrix is interchanged with the ip(i)th row. Then, the update subroutine is called, and the matrix is updated. The processes from the LU decomposition to the update subroutine are repeatedly performed until the processes are performed on all specified blocks. For the final specified block, another LU decomposition and update are performed, thereby terminating the process.

FIG. 10 shows the pseudo code for updating the remaining portions according to the information about the LU decomposition.

In the update routine shown in FIG. 10, the updating process is performed on each of the blocks A through H. For the blocks A through D and G, an exclusive subroutine is further read. Since the block I has already been updated when the LU decomposition is carried out, the subroutine is not further prepared here.

When the update routine terminates for the blocks A through D and G, the barrier synchronization is attained. Then, if the number of the processor (thread number) is 1, then the 'update 1 of e' is performed. This process is

performed by the e-update1 subroutine. After the process, the barrier synchronization is attained.

'len' indicates the width of a block to be processed in one thread. 'is' indicates the first position of the block to be processed, and 'ie' is the last location of the block to be processed. 'df-update' indicates the subroutine for updating the blocks D and F. If the blocks D and F have been updated, then the first position of a block with a block width added is stored as the first position (nbase2) of a new block, 'len' is newly computed, the first and last positions of the blocks 'is2' and 'ie2' is newly computed, and D and F are updated by df-update, thereby attaining the barrier synchronization.

Additionally, as for 'update 2 of e', when the thread number is 1, the update subroutine e-update2 of the block E is called for and barrier synchronization is performed. Similarly, as described above, 'len', 'is', and 'ie' are computed, the update routine bh-update is called for the blocks B and H, 'nbase2' is obtained, 'len', 'is2', and 'ie2' are obtained, and the process is performed by bh-update again, thereby attaining the barrier synchronization.

Furthermore, when the thread number is 1, the process is performed by u-update3 as 'update 3 of e', thereby attaining the barrier synchronization.

Afterwards, to restore the state of interchanged pivots to the original state, 'len', 'is', and 'ie' are computed, then columns are interchanged by the subroutine 'exchange', the barrier synchronization is attained, and the thread is deleted, thereby terminating the process.

FIG. 11 shows the pseudo code of the update subroutine of the blocks B and D.

In updating the block B, the subroutine b-update accesses a shared matrix a(k,n), and 'len', 'is1', and 'ie1' having the same meanings as the explanation above are computed. 'iof' indicates the number of the starting column of the block B. Then, using the matrix TRU-U having the diagonal element of the upper triangular matrix set to 1, the block B of the array a is updated by the equation shown in FIG. 11. The expression 'is : ie' indicates the process from the matrix element 'is' to 'ie'.

In updating the block D, the subroutine d-update computes a similar parameter, and updates the matrix a by the equation shown in FIG. 11 according to the lower triangular matrix TRL in the

block E.

FIG. 12 shows the pseudo code of the update subroutine for the blocks C and A.

In the update subroutine c-update for the block C, the block C is updated by the multiplication of the blocks B and F. a (1 : iof, is2 : ie2) indicates the block C, a (1 : iof, iof + 1 : iof + blk) indicates the block B, a (iof + 1 : iof + blk, is2 : ie2) indicates the block F.

In the update subroutine a-update of the block A, the block A is updated using the blocks B and D. a (1 : iof, is2 : ie2) indicates the block A, a (1 : iof, iof + 1 : iof + blk) indicates the block B, a (iof + 1 : iof + blk, is2 : ie2) indicates the block D.

FIG. 13 shows the pseudo code indicating the first and second update of the blocks G and E.

In the update subroutine a-update of the block G, as in the above mentioned subroutines, 'len', 'is2', 'ie2', 'iof', etc. indicating the width, the starting position, the ending position, etc. of a block are computed, and the block G is updated using the blocks D and H. a (iof + 1 : n, is2 : ie2) indicates the block G, a (iof + 1 : n, iof + 1 : iof + blk) indicates the block H, and a (oif +

1 : iof + blk, is2 : ie2) indicates the block D.

In the first update subroutine e-update1 of the block E, the triangular matrix above the diagonal elements of E is updated using the column vector s (1 : i, i) before the diagonal elements and the row vector (i, i + 1 : blk) after the diagonal elements.

In the second update subroutine e-update2 of the block E, the diagonal element of the upper triangular matrix of the block E is updated into the value obtained by dividing the element value before the update by the diagonal element value, updated using the row vector s (i, 1 : i - 1) before the diagonal element and the column vector s (i + 1 : blk, i) after the diagonal element, updated into the value obtained by dividing the element value of the lower triangular matrix of the block E by the value obtained by changing the sign of the diagonal element, and the diagonal element is updated into the reciprocal of the diagonal element of the block E.

FIG. 14 shows the pseudo code of the final update for the block E, and the update subroutine for the blocks D and F.

In the final update subroutine e-update3 of

the block E, the upper triangular matrix of the block E is updated by the column vector s (1 : i - 1, i) before the diagonal element and the row vector s (i, 1 : i - 1), and is updated by multiplying the element before the diagonal element of the block E by the diagonal element before update.

In the update subroutine df-update for the blocks D and F, if the width len of a block is smaller than 10, the block D or F (depending on the argument 'is' or 'ie' of the subroutine) is updated by the element value s (1 : i - 1, i) and its own row vector a (i, is : ie). The element value of the block D or F is expressed by a (1 : i - 1, is : ie) so that when the subroutine reads a matrix element, a read position is offset by the above mentioned nbase, thereby studying the block D or F by computing a column number for the element values 1 ~ i - 1. When 'len' is 20 or larger and 32 or smaller, len1 and len2 are defined, df-update id recursively called, the process shown in FIG. 14 is performed, and df-update is further called, thereby terminating the process.

FIG. 15 shows the pseudo code of the update subroutine of the blocks B and H.

In FIG. 15, bh-update performs the updating process by the operation shown in FIG. 15 when 'len' is smaller than 10. If 'len' is 20 or larger and 32 or smaller, then len1 and len2 are defined. Otherwise, len1 and len2 are separately defined, bh-update is called, the operation is performed by the equation shown in FIG. 15, and bh-update is further called, thereby terminating the process.

According to an embodiment of the present invention, for the same function (another method of obtaining an inverse matrix after the LU decomposition), as compared with the function of the numeric operation library SUN Performance library of SUN, the process can be performed using 7 CPUs at a speed 6.6 times higher.

FIGS. 16 through 29 are flowcharts of the processes of pseudo codes.

FIG. 16 shows the main process. First, the arithmetic of the inverse matrix starts with the input of the subroutine shared array A(k, n). In step S10, a thread is generated, and a total number of threads is set in the local area numthr for each thread, and a thread number assigned to each thread is set in nothrd. Additionally, for each thread, the block width is set in iblk, nb=(n+iblk-1)/iblk

is set, and i=1 is set. In step S11, it is determined whether or not i equal to nb-1. If the determination in step S11 is YES, control is passed to step S17. If the determination in step S11 is NO, nbase is set to (i-1)xiblk in step S12, A(nbase+1:n, nbase+1:nbase+n) is LU-decomposed by block width iblk, and a row block and a lower right square matrix are updated in step S13. At this time, ip(nbase+1:nbase+iblk) contains the information for exchange of rows, which is set as a subroutine. The parallel algorithm of this portion is described in the patent application previously filed by the applicant the present invention. In step S14, the subroutine exchgrow is called, and row vectors are exchanged in A(nbase+1:nbase+iblk, 1:nbase). That is, j is changed from nbase+1 to nbase+iblk, and A(j, 1:nbase) is exchanged for A(ip(j), 1:nbase) when ip (j) > j is satisfied. In step S15, the subroutine update is called to update another block. In step S16, i=i+1 is set, and control is returned to step S11. In step S17, nbase=(nb-1)xiblks is computed. In step S18, the LU-decomposition is performed on A(nbase+1:n, nbase+1:n). The information about the exchange of rows is contained in ip(nbase+1:n-1). In step S19, the subroutine

exchgrow is called, and the row vectors in A(nbase+1:n-1, 1:nbase) are exchanged. In step S20, the subroutine update is called for update. In step S21, the subroutine exchgcol is called to exchange row vectors. That is, j is changed from n sequentially by 1. When jp(j)>j is satisfied, A(1:n,j) is exchanged for A(1:n, ip(j)). In step S22, the thread generated for parallel process is deleted.

FIGS. 17 and 18 are flowcharts of the subroutine update.

In step S 30, when the subroutines b-update, d-update, c-update, a-update, and g-update are called, nbase, iblk, numthrd which is the information about the array A and the threshold, and nothrd which refers to a thread number are assigned. The subroutine b-update is called to update the block. In step S31, the subroutine d-update is called to update the block. In step S32, the subroutine c-update is called to update the block. In step S33, the subroutine a-update is called to update the block. In step S34, the subroutine g-update is called to update the block.

In step S35, the barrier synchronization is taken between threads. In step S36, it is

determined whether or not the thread has the value of nothrd of 1. If the determination in step S36 is NO, control is passed to step S38. If the determination in step S36 is YES, then the subroutine e-update is called to update the block e in step S37, and control is passed to step S38. In step S38, barrier synchronization is taken between threads. In step S39, the df-update determines and assigns the starting point (is) and the ending point (ie) to be shared by each thread. Len=(nbase+numthrd-1)/numthrd, is=(nothrd-1)xlen+1,ie=min(nbase,nothrdxlen) are computed. The first dimension leader of the block is set as istart=nbase+1, and the block width is set as len=iblk. In step S40, the subroutine df-update is called to update the block f.

In step S41, the starting point and the ending point is similarly computed and assigned. Nbase2=nbase+iblk,len=(n-nbase2+numthrd-1)/numthrd, is2=nbase2+(nothrd-1)xlen+1, ie2=min(n,nbase2+nothrdxlen) are computed. In step S42, the first dimension leader of the block is set as istart=nbase+1, and the block width is set as len=iblk. The subroutine df-update is called to update the block f. In step S43, barrier

synchronization is taken between threads. It is determined in step S44 whether or not the thread has the value of nothrd of 1. If the determination in step S44 is NO, control is passed to step S46. If the determination in step S44 is YES, then the subroutine e-update2 is called to update the block e in step S45, and control is passed to step S46. In step S46, barrier synchronization is taken between threads. In step S47, the starting point and the ending point to be assigned to each thread of the bh-update are computed. That is, len=(nbase+numthrd-1)/numthrd, is=(nothrd-1)xlen+1,ie=min(nbase,nothrdxlen) are computed. The first dimension leader of the block is set as istart=nbase+1, and the block width is set as len=iblk. The subroutine bh-update is called to update the block b. In step S48, the starting point and the ending point to be assigned to each thread of the bh-update are computed.

In step S49, nbase2=nbase+iblk, len=(n-nbase2+numthrd-1)/numthrd, is2=nbase2+(nothrd-1)xlen+1, ie2=min(n,nbase2+nothrdxnothrdxlen) are computed, the first dimension leader of the block is set as istart=nbase+1, and the block width is set as len=iblk. In step S50, the starting point

and the ending point to be assigned to each thread of the bh-update are computed. In step S51, barrier synchronization is taken between threads. In step S52, it is determined whether or not the thread has the value of nothrd of 1. If the determination in step S52 is NO, control is passed to step S54. If the determination is YES, then the subroutine e-update3 is called to update the block e in step S53, and control is passed to step S54. In step S54, barrier synchronization is taken between threads.

FIG. 19 is a flowchart of the processes of the subroutines b-update and d-update.

In the subroutine b-update, in step S60, the starting point and the ending point of the first dimension shared by each thread are computed. That is, len=(nbase+numthrd-1)/numthrd, is1=(nothrd-1)xlen+1, ie1=min(nbase,nothrdxlen), iof=nbase are computed. In step S61, A (is1:ie1,iof+1:iof+iblk) =A (is1:ie1,iof+1:iof+iblk)xTRU-U $(A(iof+1:iof+iblk,iof+1:iof+iblk))^{-1}$ is computed. TRU-U is an upper triangular matrix having the diagonal element of 1.0. Then, the subroutine exits.

In the subroutine d-update, in step S65, the starting point and the ending point to be shared by each thread are computed. That is,

len=(nbase+numthrd-1)/numthrd, is1=(nothrd-1)xlen+1, ie1=min(nbase,nothrdxlen), iof=nbase are computed. In step S66, A (iof+1:iof+iblk,is1:ie1)=TRL (A (iof+1:iof+iblk,iof+1:iof+iblk))$^{-1}$xA (iof+1:iof+iblk,is1:ie1) is computed. The TRL refers to a lower triangular matrix of the square matrix. Then, the subroutine exits.

FIG. 20 is a flowchart of the subroutine c-update.

In step S70, the starting point and the ending point of the second dimension shared by each thread are computed. That is, nbase2=nbase+iblk, len=(n-nbase2+numthrd-1)/numthrd, is2=nbase2+(nothrd-1)xlen+1, ie2=min(n,nbase2+nothrdxlen), iof=nbase are computed. In step S71, A(1:iof,is1:ie2)=A(1:iof,iof+1:iof+i1bk)xA(iof+1:iof+ilbk,is2:ie2) is computed. Then, the subroutine exits.

FIG. 21 is a flowchart of the subroutine a-update.

In step S75, the starting point and the ending point of the second dimension shared by each thread are computed. That is, len=(nbase+numthrd-1)/numthrd, is2=(nothrd-1)xlen+1, ie2=min(nbase,nothrdxlen), iof=nbase are computed.

In step S76, A (1:iof,is2:ie2)=A (1:iof,is2:ie2)-A(1:iof,iof+1:iof+iblk) xA(iof+1:iof+iblk,is2:ie2) is computed. Then, the subroutine exits.

FIG. 22 is a flowchart of the subroutine g-update.

In step S80, the starting point and the ending point of the second dimension shared by each thread are computed. That is, len=(nbase+numthrd-1)/numthrd, is2=(nothrd-1)xlen+1, ie2=min(nbase,nothrdxlen), iof=nbase are computed. In step S81, A (iof+1:n,is2:ie2) =A (iof+1:n,is2:ie2) -A (iof+1:n,iof+1:iof+iblk) xA (iof+1:iof+iblk,is2:ie2). Then, the subroutine exits.

FIG. 23 is a flowchart of the subroutine e-update.

In this routine, the argument is assigned such that the element of the upper left corner of the block E can be the first element of S (k, *). In step S85, i=1 is set. In step S86, it is determined whether or not i>iblk. If the determination in step S86 is YES, the subroutine exits. If the determination in step S86 is YES, s(1:i-1,i+1:iblk)=s(1:i-1,i+1:iblk)-s(1:i-1,i)×s(i,i+1:iblk), i=i+1 are computed in step S87

and the subroutine exits.

FIG. 24 is a flowchart of the subroutine e2-update.

First, the argument is assigned such that the element of the upper left corner of the block E can be the first element of S (k, *). In step S90, i=1 is set. In step S91, it is determined whether or not i>iblk. If the determination in step S91 is YES, the subroutine exits. If the determination in step S91 is NO, tmp=1.0/S(i,i), S(i,i:i-1)=tmpxS(i,1:i-1), S(i+1:iblk,1:i-1)=S(i+1:iblk,1:i-1)-S(i,1:i-1)xS(i+1:iblk,i), S(i+1:iblk,i)=-A(i+1,iblk,i)xtmp, A(i,i)=tmp, i=i+1 is computed in step S92, and control is returned to step S91.

FIG. 25 is a flowchart of the subroutine e3-update.

First, the argument is assigned such that the element of the upper left corner of the block E can be the first element of S (k, *). In step S95, i=1 is set. In step S96, it is determined whether or not i>iblk. If the determination in step S96 is YES, the subroutine exits. If the determination in step S96 is NO, S(1:i-1,1:i-1)=S(1:i-1,1:i-1)-S(1:i-1,i)xS(i,1:i-1), S(1:i-1,i)=S(1:i-1)xS(i,i), i=i+1 are computed in step S97, and control is returned

to step S96.

FIG. 26 is a flowchart of the subroutine df-update.

The subroutine is a recursive program. First, is and ie are assigned to the starting point and the ending point indicating the range of the process in each thread. The argument is assigned such that the element of the upper left corner of the block E can be the first element of S (k, *). The first dimension leader istart of the block to be processed and the block width len are assigned. In step S100, it is determined whether or not len<10 is held. If the determination in step S100 is NO, control is passed to step S104. If the determination in step S100 is YES, then i=1 is set in step S101, and it is determined in step S102 whether or not 1<=len is held. If the determination in step S102 is NO, the subroutine exits. If the determination in step S102 is YES, then js=istart, je=istart-1+len-1, A (js:je,is,ie)=A (js:je,is,ie)-S (js:je,i) xA (istart+i-1,is:ie) are computed in step S103, and control is returned to step S102.

In step S104, it is determined which is held, len>=32 or len<=20. If the determination in step S104 is NO, then len1=len/3, len2=len-len1 are set,

and control is passed to step S107. If the determination in step S104 is YES, then len1=len/2, len2=len-len1 are set in step S105, and control is passed to step S107. In step S107, the subroutine bf-update is recursively called (the block leader istart and the block width len1 are assigned). In step S108, js=istart, je=istart+len1-1, js2=js-nbase, je2=js2+len1-1, js3=js2+len1, je3=js3+len2-1, js4=istart+len1, je4=js4+len2-1, A(js:je,is:ie)=A(js:je,is:ie)-S(js2:je2,js3:je3)xA(js4,je4:len,is:ie), istart2=istart+len1 are computed, and control is passed to step S109. In step S109, the subroutine df-update is recursively called (the block leader istart2 and the block width len2 are assigned), and the subroutine exits.

FIG. 27 is a flowchart of the subroutine bh-update.

The subroutine is a recursive program. First, is and ie are assigned to the starting point and the ending point indicating the range of the process in each thread. The argument is assigned such that the element of the upper left corner of the block E can be the first element of S (k, *). The first dimension leader istart of the block to

be processed and the block width len are assigned. In step S105, it is determined whether or not len<10 can be held. If the determination in step S115 is NO, then control is passed to step S119. If the determination in step S115 is YES, then i=1 is set in step S116, and it is determined in step S117 whether or not i<=len is held. If the determination in step S117 is NO, then the subroutine exits. If the determination in step S117 is YES, then j=istart+i-1, j2=j-nbase, je=istart+len-1, je2=je-nbase, A (is:ie,j) =-A (is:ie,j) xS (j2,j2), A (is:ie,j) =A (is:ie,j) -A (is:ie,j+1:je) xS (j2+1:je2,j2) are computed in step S18, and control is passed to step S117.

In step S119, it is determined which is held len>=32 or len<=20. If the determination in step S119 is NO, control is passed to step S121. If the determination is YES, control is passed to step S120. In step S120, len1=len/3, len2=len-len1 are computed, and control is passed to step S122. In step S121, len1=len/3, len2=len-len1 are computed, and control is passed to step S122. In step S122, the subroutine bh-update is recursively called, the target block leader istart and the block width len1 are assigned. In step S123, js=istart,

je=istart+len1-1, js2=js-nbase, je2=js2+len1-1, js3=istart+len1, je3=js2+len2-1, js4=js3-nbase, je4=js4+len2-1, A (is:ie,js:je) =A (is:ie,js:je) -A (is:ie,js3:je3) xS (js4:jse,js2:je2), istart2=istart+len1 are computed. In step S124, the subroutine bh-update is recursively called, and the target block leader istart2 and the block width len1 are assigned. Then, the subroutine exits.

FIG. 28 is a flowchart of the subroutine exchgrow.

In step S130, barrier synchronization is taken. In step S131, the starting point and the ending point to be shared by each thread are computed. That is, len=(nbase+numthrd-1)/numthrd, is=(nothrd-1)xlen+1, ie=min(nbase,nothrdxlen), j=nbase+1 are computed. In step S132, it is determined whether or not j>min(n-1,nbase+iblks) is held. If the determination is YES, then control is passed to step S136. If the determination is NO, then control is passed to step S133. In step S133, it is determined whether or not jp(j)>j is held. If the determination in step S133 is NO, then control is passed to step S135. If the determination in step S133 is YES, then A (j,is:ie) is exchanged for A (ip(j),is:ie) in step S134, and control is passed

to step S135. In step S135, j=j+1 is computed, and control is returned to step S132. In step S136, barrier synchronization is taken, and the subroutine exits.

FIG. 29 is a flowchart of the subroutine exchgcol.

In step S140, barrier synchronization is taken. In step S141, the starting point and the ending point to be shared by each thread are computed. That is, len=(nbase+numthrd-1)/numthrd, is=(nothrd-1)xlen+1, ie=min(nbase,nothrdxlen), j=n-1 are computed, and control is passed to step S142. In step S142, it is determined whether or not j<1 is held. If the determination in step S142 is YES, control is passed to step S146. If the determination in step S142 is NO, then it is determined whether or not jp(j)>j is held in step S143. If the determination in step S143 is NO, control is passed to step S145. If the determination in step S143 is YES, then A (is:ie,j) is exchanged for A (is:ie,jp(j)) in step S144, and control is passed to step S145. In step S145, j=j-1 is set, and control is returned to step S142. In step S146, barrier synchronization is taken, and the subroutine exits.

Refer to the following textbook for the common algorithm of matrix computation.

G. H. Golub and C. F. Van Loan "Matrix Computations" The Johns Hopkins University Press, Third edition 1996

According to the present invention, a method of solving an inverse matrix can be realized with high performance and scalability.